ARL-TR-8167 • Sep 2017

ARL
US Army Research Laboratory

Ego-motion and Tracking for Continuous Object Learning: A Brief Survey

by Jason Owens and Philip Osteen

ARL-TR-8167 ● SEP 2017

US Army Research Laboratory

Ego-motion and Tracking for Continuous Object Learning: A Brief Survey

by Jason Owens and Philip Osteen
Vehicle Technology Directorate, ARL

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* September 2017	**2. REPORT TYPE** Technical Report	**3. DATES COVERED (From - To)** October 2016-September 2017
4. TITLE AND SUBTITLE Ego-motion and Tracking for Continuous Object Learning: A Brief Survey		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Jason Owens and Philip Osteen		**5d. PROJECT NUMBER** APPLE
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) US Army Research Laboratory ATTN: RDRL-VTA Aberdeen Proving Ground, MD 21005-5066		**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-8167
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
Primary author's email: <jason.l.owens.civ@mail.mil>.

14. ABSTRACT
This report provides a brief review of current and past research related to the tasks of ego-motion estimation and object tracking from the viewpoint of their role in continuous object learning.

15. SUBJECT TERMS
intelligent systems, perception, incremental learning, instance recognition, ego-motion estimation, object tracking

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Jason Owens
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	50	**19b. TELEPHONE NUMBER (Include area code)** 410-278-7023

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

1. Introduction

Ego-motion estimation, or the ability to estimate one's own motion, is important for an embodied agent (whether animal or robot) to know where it is and how it may interact with the environment. Object tracking, on the other hand, complements ego-motion awareness by estimating the independent motion of *other* objects in the environment. In many ways, they provide a level of continuity to our perception and actions that can enable us as humans to build models of the world and the objects within it, and should do likewise for robots.

This survey is intended to broadly describe the current state of the ego-motion estimation and object tracking fields. We believe these capabilities are required to support online adaptive learning of objects in dynamic environments. This is by no means a complete treatment of these topics; rather, this is a broad review meant to focus on the current capabilities in these areas and point out the challenges with respect to a continuous object learning system.

1.1 Continuous Object Learning

Our long-term goal is to identify the theory and principles underlying real-world continuous and adaptive visual perception for intelligent systems. We believe that it is a requirement that embodied intelligent systems extract a significant portion of their knowledge directly from experience in the operating environment.

We hypothesize that an agent that continuously learns to recognize objects from experience (a continuous object learner [COL]) requires mechanisms to (1) track ego- and object-motions for object permanence and multiview object modeling, (2) recognize and focus attention on things and stuff of interest, (3) convert raw sensor data into rich, flexible internal representations, (4) detect previously observed object instances and object categories, (5) detect novel objects, and (6) generalize collections of novel instances into categories over time.

Our hypothesis is formed through reasoning from the evidence available in human perception and our experience with existing computational perception algorithms. In Fig. 1, we present a high-level data flow diagram for a hypothesized continuous object learner.

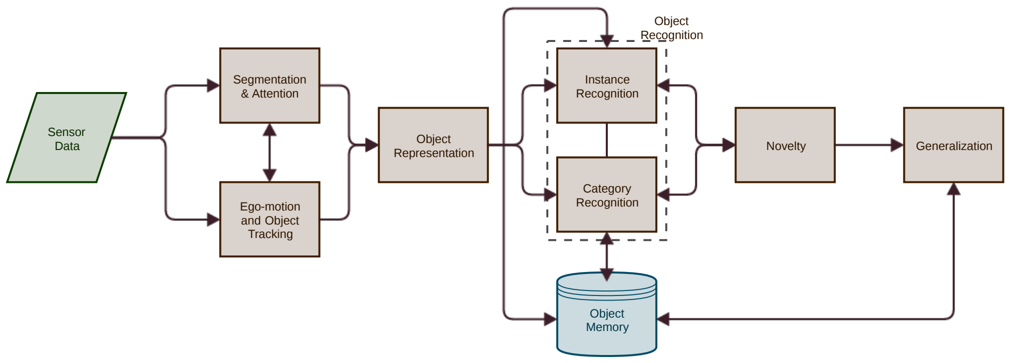


Fig. 1 A notional high-level system diagram for a continuous object learner

If we do not wish to arbitrarily ignore agent and object motion, which is a requirement for mobile agents and dynamic environments (the purview of Army robotics), then the system must have a mechanism to compute the motion of the sensors with respect to the environment as well as object motions. These capabilities are required for various aspects of robot operation, including data association between successive frames of input. Continuity of perception is critical for humans in determining object permanence and identity, and the phenomenon depends on dynamically adaptive ego-motion estimation functions of the human perceptual system.[1] If humans (or a COL) could not relate what we see at time t to time $t+1$, it would be much harder to learn from experience. If the COL were unable to relate images of objects in successive frames, it would be much more difficult to associate multiple views of the same object; instead, each view would be considered a new object. Furthermore, it is easy to see how the agent's motion with respect to the world and independent object motions are related and intertwined: any ego- or object-motion observed by a sensor will result in changes to the raw data between frames. Without some mechanism to explain the change and relate it to the previous frame, we can make no inference about the change in the state of the world. In addition, object motion is related to the segmentation and attention problem through the fact that an independently moving object automatically provides segmentation cues, related to the Gestalt principle of common fate.[2]

1.2 Ego-Motion and Tracking

Mobile systems operating in complex environments need some way of relating frames of data observed at different times to one another. Ego-motion estimation is the process of tracking sensor motion to determine the current motion and pose of the agent within an environment. Typical ego-motion algorithms measure static aspects of the environment; however, we require robots to also function in complex, dynamic environments. For a COL, moving objects in the environment are interesting for several reasons: we want to avoid colliding with another moving entity (in most cases), a moving object provides automatic segmentation for itself by virtue of its change with respect to the environment, and we would like a COL to identify what these moving things are to potentially interact with them. We therefore expect complementary object tracking and ego-motion algorithms to provide a clear estimate of the geometry and dynamics of the world, and the robot's pose within the environment.

One of the simplest ways to estimate motion for a basic wheeled robot (which can be generalized to nonwheeled systems) is to make use of wheel odometry (derived from the Greek *odos* for "route" and *metron* for "measurement"). Knowing the size of the wheels, where they are placed on the robot and how they are turning provides enough information to compute a motion estimate. While this is a common and widely used source of estimated motion, it is usually not the only one since it is subject to many errors from wheel slip or skidding, and only provides a 3-dimensional (3D) estimate (x, y, θ) based on contact with the ground. To help mitigate some of these drawbacks, an inertial measurement unit (IMU) may be used to get more locally accurate information about the 3D rotational velocity and 3D acceleration of the robot in order to integrate position and velocity over time.

The method of motion estimation we just discussed focuses on 2 sources of *proprioceptive* information. Proprioceptive sensors record information that originates from within an agent (e.g., recording the wheels' motion as driven by the motors but **not** how the wheels interact with the ground, or the inertial sensing of the IMU that gives a change in the rotational velocity). In contrast, sensors like the accelerometers of the IMU, cameras, and laser range finders directly sense some aspect of the surrounding environment: accelerometers sense the force of gravity, cameras sense light energy from the environment, and laser range finders directly measure distance to sufficiently reflective objects in the environment. These sensors are *exteroceptive*, and besides providing potential improvements to motion estimation (e.g., measuring perceived motion with respect to fixed external landmarks) they also open up the possibility of measuring and modeling the environment.

Using cameras for motion estimation is a form of odometry, called visual odometry (VO) in the literature. The task of visual odometry is simply to measure the motion of the agent using visual sensors. However, if the agent simultaneously builds a map using exteroceptive information from the cameras and laser sensors, then the task is called simultaneous localization and mapping (SLAM). The benefit of such an algorithm is that a robot is able to recognize that it has been in a particular place before, since the motion estimation, map, and localization allow it to maintain and recognize features of the environment.

If there are objects moving around the robot, however, odometry based on exteroceptive input will fail unless the moving objects are ignored or the states of the moving entities are correctly modeled. Therefore, most VO and SLAM algorithms ignore dynamic information. However, handling dynamic environments is important for continuous object learners, and we can model object motion using tracking algorithms, whose goal is to maintain awareness of specified objects in an arbitrary environment. Typically, trackers require the target object's location in an initial frame, then they must find instances of the object in subsequent frames. In general, the solution is improved by estimating the motion of the tracked object, and this is in turn aided by also knowing the motion of the tracking sensor as well as the model of the static environment. The tracking problem places unique constraints on algorithms compared to other computer vision problems such as object detection. A tracking algorithm must run at real-time speeds to be useful on a live mission, should support scaling object window sizes dynamically, and should be able to track potentially deformable objects using a very small set of training data (e.g., one or more example frames).

A critical component of any tracking algorithm is determining which objects should be tracked in the first place. Generally, tracking approaches use 1 of 3 methods to make this determination. *Motion-based tracking* assumes that moving objects can be segmented from the environment, and that only dynamic objects are worth tracking. In many cases, this is a reasonable assumption that represents many of the use cases for tracking while simplifying the segmentation problem. However, for a moving sensor, this approach is more challenging since egomotion must also be estimated in order to separate dynamic objects from a static background. Furthermore, one can imagine situations wherein mobile platforms need to track (temporarily) stationary targets, while they themselves maneuver through the environment.

Model-based tracking approaches support tracking static and dynamic targets without the need for a user to explicitly specify the target in the environment, instead using a priori models of each target. This advantage comes at the cost of generality, however, as such systems only support tracking a finite set of objects that are specified ahead of time. Although we envision tracking as a more primitive process for a COL, and do not assume we have a priori models available, we will briefly review existing model-based methods for completeness. *Model-free tracking*, in contrast, does not require a priori models to specify which objects to track, which means that some other method must supply the set of tracking objects. In practice, this often comes from user-supplied bounding boxes in the first frame of an input sequence. The model must then be built on the fly, using just the data from the initial frame for training, and must also dynamically adapt its model as more instances are observed. If no explicit user-supplied object segmentation is given, another segmentation method must be used in its place. In this case, either object proposals[3] or motion-based segmentation algorithms[4] are employed. For the purposes of this discussion, we group model-free trackers based on motion segmentation with other motion-based trackers.

2. Brief Survey: Ego-motion

In this section, we review the main approaches to ego-motion estimation based on visual input. VO computes pose based on visual input, while SLAM often makes use of visual odometry for local pose estimation, but also constructs maps and recognizes places to improve pose estimation over time. We follow the surveys of VO and SLAM with a breakdown of attributes for distinguishing selected algorithm capabilities, and a discussion of the challenges we may face when integrating ego-motion estimation into a continuous object learner.

2.1 Visual Odometry

Visual odometry is concerned with computing the motion of a sensor using visual data streaming from that sensor. Following the classification of Engel et al.,[5] the computational method for incremental pose transformation between sensor frames can be described using 2 dimensions: the nature of the input and the density of the input. The first dimension describes what *values* are used in the computation; in visual odometry, the data provided by the sensor(s) are spatially organized photometric measurements of the amount of light registered by the sensor (i.e., the pixel values). If an algorithm uses this photometric information, then it is called a *direct* method. Alternatively, if there are derived values computed from the image, (e.g., keypoints, descriptors, and/or vector flow fields), then we call the method *indirect*. Note that since direct methods use the photometric information directly, the optimization models photometric noise, while the derived values in the indirect case are geometric in nature (points and vectors), and therefore the optimization models geometric noise.

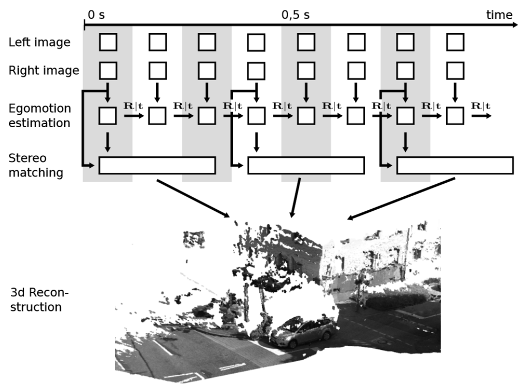
The second dimension indicates how much information is used for the computation. If an algorithm attempts to use all the pixels (or as many as possible) from the input, then it is called a *dense* method. In contrast, a *sparse* method specifically uses a small subset of the pixels/points available, usually around 2 orders of magnitude less than than the number of pixels. Dense methods do not extract a discrete set of features, but instead rely directly on the data itself to compute an estimate of motion. The main idea is to use more of the image data than sparse methods with the goal of improving the camera and environment model motion estimates. In contrast, sparse methods often use discrete feature keypoints (e.g., scale-invariant feature transform [SIFT],[6] oriented FAST and rotated BRIEF [ORB][7]). These keypoints, along with the data, serve as input to an algorithm that will produce a set of descriptors. The descriptors are representations of the input, designed to enable feature matching between frames by comparing the distance between pairs of descriptors.

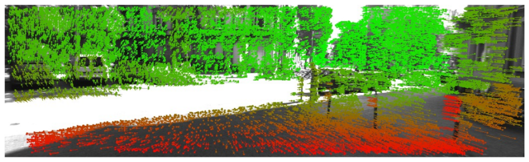
For simplicity, we primarily focus on stereo, red, green, blue and depth (RGB-D), and laser-based modalities that can directly compute 3D features within the environment, as they are the most useful for enabling accurate, scale drift-free maps and motion estimates. Monocular algorithms, while powerful and efficient, cannot compute the scale of the environment (although they can be filtered with other odometry methods that can). 2-dimensional (2D) laser scan-matching algorithms, while very popular on experimental robotic systems, are also insufficient for our needs, since they cannot compute full 6-DOF pose estimates and are easily confused by nonflat environments.

`libviso2` by Geiger et al.[8] is a simple but effective stereo visual odometry algorithm created by the group responsible for the KITTI benchmark suite.[9] It is a prime example of sparse indirect visual odometry methods (see Scaramuzza and Fraundorfer[10] for a nice overview of the approach). Ego-motion is computed using simple blob and corner-like features distributed over the full image, which are stereo matched between the left and right frame to compute 3D pose, and then temporally matched between successive frames to estimate motion (Fig. 2). Complex feature descriptors are not needed since there is an assumption that frames are temporally and spatially close (i.e., from a 30-Hz camera). Instead, a simple sum of absolute differences of an 11×11 pixel block around the keypoint is used as a descriptor distance method. Ego-motion estimation is implemented as a Gauss-Newton minimization of reprojection error:

$$\sum_{i=0}^{N} \left\| \mathbf{x_i}^{(l)} - \pi^{(l)}(\mathbf{X_i}, \mathbf{r}, \mathbf{t}) \right\|^2 + \left\| \mathbf{x_i}^{(r)} - \pi^{(r)}(\mathbf{X_i}, \mathbf{r}, \mathbf{t}) \right\|^2 , \tag{1}$$

where $\mathbf{X_i}$ are the 3D points corresponding to the features $\mathbf{x_i^{(l)}}, \mathbf{x_i^{(r)}}$ in the left and right images, $\pi^{(l)}, \pi^{(r)}$ are the corresponding left and right projection functions, based on intrinsic and extrinsic calibration of the cameras, and $\mathbf{r}, \mathbf{t}$ are the rotation and translation parameters being estimated.




(a) Feature matching (2 frames, moving camera)

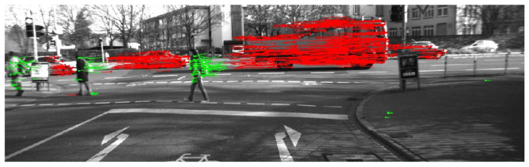
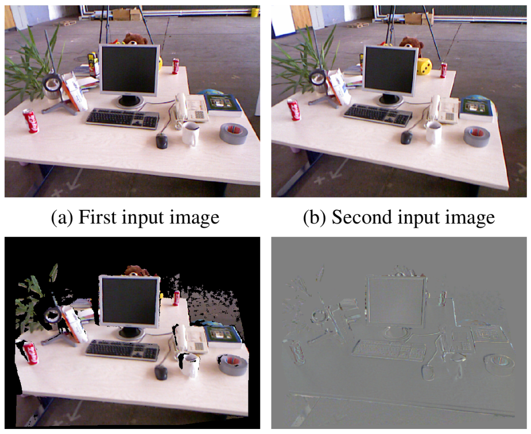

(b) Feature tracking (5 frames, static camera)

Fig. 2 `libviso2` visual odometry system, an example of a sparse, indirect VO method. The left image illustrates the overall operation of the system (temporal ego-motion estimation, stereo matching and 3D reconstruction). The right image shows the matched features and their motion in 2 situations: (a) moving camera and (b) static camera. Reproduced from Geiger et al.[8]

Steinbrucker et al.[11] present Dense Visual Odometry (DVO), a VO algorithm that is based on RGB-D sensors instead of stereo cameras. DVO eschews feature extraction in favor of a dense model in order to take advantage of as much of the image as possible. The algorithm utilizes a photometric consistency assumption to maximize the photoconsistency between 2 frames. Photometric consistency means that 2 images of the same scene from slightly different views should have the same measurements for pixels corresponding to a given 3D point in the scene. Steinbrucker et al. minimize the least-squares photometric error:

$$E(\xi) = \int_{\Omega} \left[I(w_\xi(x, t_1), t_1) - I(w_\xi(x, t_0), t_0) \right]^2 dx, \tag{2}$$

where ξ is the twist representing the estimated transformation, and the $w(x,t)$ function warps the image point x to a new image point at time t using the transformation ξ and the associated depth map. $I(x', t)$ is a function that returns the image intensity value at position x' at time t. Note how this represents a dense direct approach, integrating over the entire image domain (Fig. 3), while the sparse indirect approach in `libviso2` is a sum over only the extracted features. The 3D points in an RGB-D approach come directly from the produced depth map and do not require any explicit stereo computations. In practice, dense approaches vary in their actual density: RGB-D based dense approaches can be much more dense than monocular or stereo approaches, although in both cases they use more image data than a sparse approach. Unfortunately, the cost of higher density is more computation, so the tradeoff often depends on the accuracy required in the motion estimation.


(a) First input image (b) Second input image

(c) Warped second image (d) Difference image

Fig. 3 An example of RGB-D image warping in DVO. (a-b) show the input images separated by time, while (c) shows the second image warped to the first, using the photometric error and the depth map to compute how the pixels should be projected into the warped frame. (d) illustrates the small error between (a) and (c). Reproduced from Steinbrucker et al.[11]

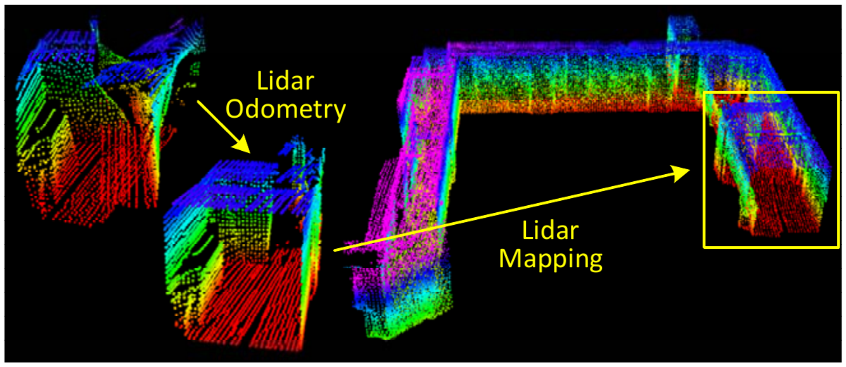
The use of current RGB-D sensors has benefits and drawbacks: as active stereo devices, they do not require strong features in the environment to compute depth values, yielding dense depth images; however, they usually have relatively short range (relying on a pattern or measured time of flight from an infrared (IR) projector) and do not work well (if at all) in outdoor conditions.*

Lidar odometry and mapping (LOAM[12]) and visual odometry-aided LOAM (V-LOAM[13]) are related ego-motion algorithms by Zhang and Singh that use 3D laser scanners (e.g., the Velodyne HDL-32[14]) to compute both maps and motion estimates (see output and system flow in Fig. 4).† LOAM works (and works very well, see the KITTI[9] odometry benchmarks) by carefully registering 3D point cloud features to estimate sensor motion at a high frame rate, and then integrating full undistorted point clouds into the map at lower frame rates while adjusting the sensor pose estimates. This algorithm can work with sweeping single-line laser scanners as well as full 3D scanners by estimating the pose transform between each line scan. If such a sensor is available, this algorithm produces some of the lowest error motion and pose estimates over large-scale paths. The primary drawback is that the code is no longer available in the public domain, since it has been commercialized as a

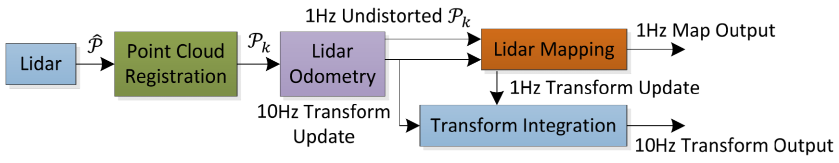
*Newer RGB-D sensors seem to address this problem, with slightly longer ranges and claims of outdoor operation. We do not yet have direct experience to comment on these claims.

†LOAM and V-LOAM are not full SLAM algorithms, since they do not include loop closure and cannot perform global localization.

stand-alone metrology product.*



(a) Example LOAM results.



(b) LOAM system diagram.

Fig. 4 An example of the LOAM mapping output along with a high-level block diagram showing the system operation. The important function is the point cloud registration, enabling the light detection and ranging (LIDAR) odometry computation that feeds the transform and mapping outputs. Reproduced from Zhang et al.[12]

Monocular visual odometry algorithms have recently become more popular on small robotic platforms due their extremely simple sensor arrangement (i.e., a single camera) and their ability to operate on mobile computing hardware.[16,17] Several recent monocular algorithms (semi-dense visual odometry,[18] semi-direct visual odometry [SVO],[19] and Direct Sparse Odometry [DSO][5]) demonstrate impressive mapping performance with low drift, all within a direct VO paradigm (i.e., they *directly* make use of the values from the camera and therefore optimize a photometric error). These algorithms operate by estimating semi-dense depth maps using strong gradients in the image along with temporal stereo (comparing 2 frames that differ in time, with the assumption that the camera was moving during the time period), and make extensive use of probabilistic inverse depth estimates. DSO is particularly

*Although the code was originally available as open source on Github and documented on the Robot Operating System (ROS) wiki,[15] the repositories have since been removed. Fortunately, some forks of the repositories still exist, but the full ROS integration does not seem to exist. In addition, the code is undocumented, has hard coded transformation assumptions, and is not written for modularity or future programmers. However, it may provide sufficient guidance to those who wish to adapt and extend the algorithm in the future.

interesting since it is a direct method using sparse samples (but no feature detection) and it exploits photometric sensor calibration to improve the robustness of the motion estimation over existing methods.

However, for monocular algorithms to be useful for typical robotics applications, there must be some way to estimate the scale of the observed features for map generation and motion control. A typical (but not singular) method for accomplishing this is by filtering the visual features with some other metric sensor such as an IMU. This can be considered a sub-field of visual odometry, typically called visual-inertial navigation. Since we assume we will have access to stereo, RGB-D, or 3D laser information, reviewing this topic is out of scope for this document. However, for more information, see Weiss' tutorial on "Dealing with Scale" presented at Computer Vision and Pattern Recognition (CVPR) 2014.[20]

2.2 SLAM

Many of the VO algorithms we discuss create local models of the environment to achieve more accurate motion estimates (i.e., odometry with less drift). It might be natural to assume these algorithms could be included as the front-end in a SLAM framework, and that assumption would be correct (see Cadena et al.[21] for a great overview of SLAM with a look toward the future). A SLAM front-end provides motion estimates (like what we get from VO), while a SLAM back-end optimizes a map given local constraints between sensor poses, observed landmarks, and global pose relationships detected as "loop closures". A loop closure is the detection of 2 or more overlapping observations, often separated considerably in time but not in space; see Fig. 5 for a simple illustration. For example, if a robot maps a room, exits to map another room or hallway, and then re-enters the first room, it should detect a loop closure by virtue of being in the same place it was before and therefore seeing the same landmarks. The term "loop" comes from the fact that the pose graph (where sensor poses are vertices, and edges represent temporal or co-observation constraints) forms a cycle or loop of edges and vertices when a revisitation occurs.

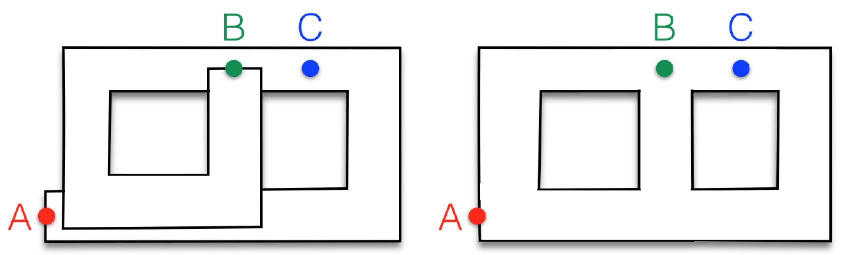


Fig. 5 An example of the loop-closing concept in a simplified hallway. The left image shows a camera that has computed visual odometry and revisited both point A and point B without closing the loop. The right image, on the other hand, is the map result from a SLAM algorithm after detecting the loop closure and optimizing the map. Reproduced from Cadena et al.[21]

Stereo LSD-SLAM[22] is a camera-based algorithm that represents an extension of the LSD-SLAM[23] monocular algorithm to a stereo setting. LSD stands for large-scale direct SLAM and, as the name indicates, represents a direct, dense image-based VO algorithm integrated into a SLAM system handling mapping and loop closure. They build on the original algorithm by adding support for static stereo to estimate depth, requiring no initialization method (typically required for monocular VO algorithms) and obviating the need for the original scale-aware similarity transform estimation. In addition, they add in a simple exposure estimation procedure to help counteract the effects of lighting changes in real-world environments. Their results are impressive but a little slow for larger image sizes (640×480 runs at only 15 Hz) if one is concerned about running in real-time on a robot. Currently, code is available for *only* the monocular version.

As an example of a hybrid-indirect VO methodology used in a SLAM framework, Henry et al.[24] was one of the first high-quality dense mapping algorithms to make use of RGB-D sensors like the Microsoft Kinect. The core algorithm is composed of a frame-frame iterative closest point (ICP) algorithm on the point cloud derived from the depth image, which is jointly optimized with sparse feature correspondences derived from the red, green, blue (RGB) images. It uses both sparse and dense data, thus forming what we call a "hybrid" methodology. The ICP and sparse feature alignment are complementary: the former performs better with significant geometry complexity in the environment, even when featureless, while the latter supports alignment when there are features, but little to no geometry (e.g., walls and floors with posters or other textured objects). To generate higher-resolution clouds, they postprocess the optimized map by integrating depth frames using a surfel-based approach[25] to better adapt regions with differing point density.

Another sparse-indirect VO method applied in a SLAM framework is ORB-SLAM2,[26] an extension of the original ORB-SLAM framework[27] to stereo and RGB-D cameras for metric scale. ORB-SLAM2 is a combination of existing techniques that provide a robust SLAM system with the rare capability of reusing maps for localization in previously visited areas. As the name implies, Mur-Artal and Tardós make use of the ORB[7] feature detector/descriptor to detect and describe features in each frame (Fig. 6). Several aspects of this algorithm make it stand out: (1) both near and far features are included in the map, (2) the map is reusable for localization without mapping, and (3) there is no dependence on a graphics processing unit (GPU). While aspect (3) has obvious benefits (i.e., it can be adapted to run on systems without a GPU), the first 2 require some additional explanation. In most recent monocular VO approaches, feature depth (within a given keyframe) is parameterized as inverse depth. This makes it easy to incorporate points at larger distances, since $\frac{1}{d}$ for infinite d is just 0. Points at infinity or with uncertain depth are still useful for computing accurate rotations, even if they cannot be used to estimate translation. By incorporating near (i.e., depth computable from stereo or RGB-D) and far points, ORB-SLAM2 can generate better pose estimates using more information. Finally, while SLAM algorithms are used to build maps, most are unable to localize within those maps without generating new map data. This feature is particularly useful for a robotic system that may encounter new areas over time, but still navigate previously mapped regions repeatedly.

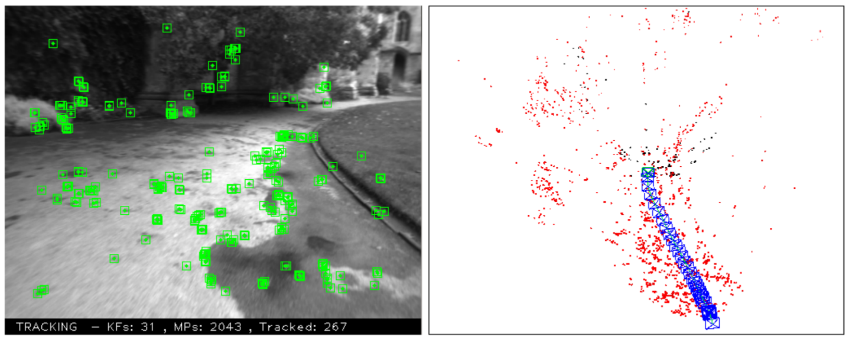
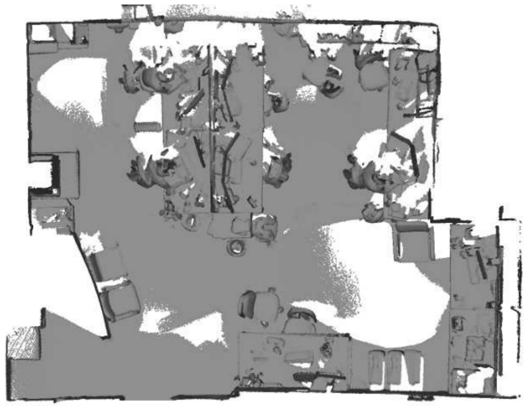


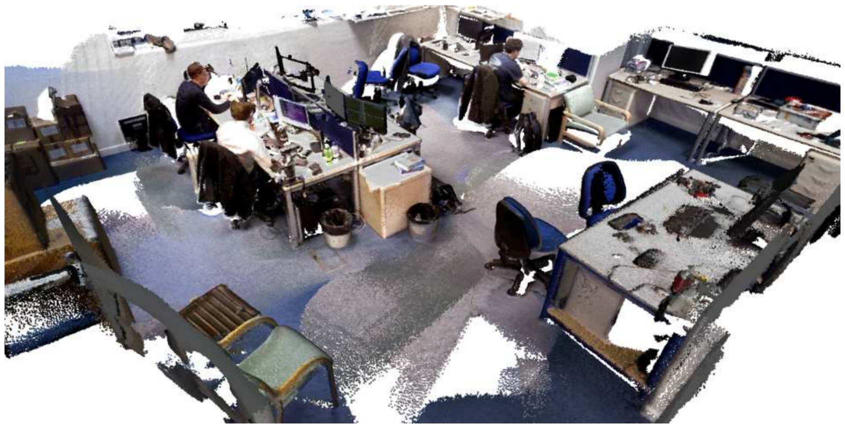
Fig. 6 ORB-SLAM feature and map density. Note the sparsity of the feature-based map. Reproduced from Mur-Artal et al.[27]

In the popular subfield of 3D reconstruction using RGB-D sensors, 2 recent methods exemplify the volumetric integration approach. Ever since the influential Kinect-Fusion papers[28,29] demonstrated the use of on-GPU truncated signed distance function (TSDF) volumes, many researchers have expanded on and refined the approach.[30–36] We highlight 2 recent examples, both capable of extended model reconstruction (i.e., a larger volume than can fit in GPU memory) and both utilizing full RGB-D frame information.

ElasticFusion[37] computes a dense surfel model of environments, notably without using pose graph optimization. Instead, the system relies on frequent registration with both active (recently acquired and currently used for pose estimation) and inactive (local but older observations) portions of the map, the latter reflecting loop closures. Upon registration with inactive portions of the map, the algorithm induces a nonrigid deformation that helps to bring that portion of the map into alignment with the currently active version. It is this deformation process that obviates the need for a global graph optimization phase, with the assumption that the registration reduces the error in the model enough to not require the optimization. Indeed, the results reported show some of the lowest reconstruction errors on popular benchmark datasets. However, it should be noted that they do not show spaces larger than a large office environment (see Fig. 7 for an example), while previous approaches such as Kintinuous[33] show larger outdoor scenes and multiple indoor floors.



(a)



(b)

Fig. 7 An example map (or model) produced using ElasticFusion.[37] 7a shows the top-down view of the environment, while 7b shows a colored point cloud highlighting the detail captured by the algorithm. This is representative of the dense methods described in the text; note the differences with the map in figure 6. Reproduced from Whelan et al.[37]

BundleFusion[38] aims to provide an all-around solution to real-time 3D reconstruction, with the unique capability of real-time frame deintegration and reintegration in the global volumetric model. A sparse to dense local pose alignment algorithm is used to improve the pose estimation (utilizing SIFT features for sparse correspondence) between the frames, while a global optimization is performed after scanning has ended. Like the previous approach, this algorithm produces very good dense scene models, but suffers from 2 main limitations: scene size (or recording time) is limited to a maximum of 20,000 frames (due the nature of their on-GPU data structures), and they require 2 powerful GPUs in parallel to achieve real-time performance, limiting application to larger robotic systems capable of carrying and powering twin GPUs.

2.3 Attributes

In this subsection, we identify salient properties of the algorithms reviewed in the egomotion survey. Table 1 compares selected algorithms according to the following attributes:

Input What is the input modality? ([Mono]cular camera, [Stereo] camera, [RGB-D], [2D] LIDAR, [3D] LIDAR)

Direct Direct versus indirect parameter estimation (i.e., photometric versus geometric optimization)

Dense Is the representation dense (versus sparse)?

Mapping Does the system generate maps?

Loop Closing Does the method handle place recognition for closing loops (revisited locations) in maps?

Known Scale Does the method product maps with known scale (related to sensor modality)?

Persistent Can the existing implementation store new maps, and load and modify previously generated maps?

GPU Does the algorithm utilize/require a GPU?

Scalability Color saturation represents a comparative and qualitative estimation of the scalability of the algorithm.

Code Do the authors make source code available?

Table 1 An overview of selected ego-motion algorithms reviewed in this section and their attributes.

	Input	Direct	Dense	Mapping	Loop Closing	Known Scale	Persistent	GPU	Scalability	Code
viso2[8]	Stereo	✗	✗	✗	✗	✓	✗	✗	🟦	✓
DVO[11]	RGB-D	✓	✓	✗	✗	✓	✗	✗	🟦	✓
LOAM[12]	2D/3D	✗	✗	✓	✗	✓	✗	✗	🟦	✓[a]
vLOAM[13]	2D/3D + Mono	✗	✗	✓	✗	✓	✗	✗	🟦	✗
SDVO[18]	Mono	✓	✓[b]	✗	✗	✗	✗	✗	🟦	✗
SVO[19]	Mono	✓[c]	✗	✗	✗	✗	✗	✗	🟦	✓
DSO[5]	Mono	✓	✗	✗	✗	✗	✗	✗	🟦	✓
SLSD-SLAM[22]	Stereo	✓	✓	✓	✓	✓	✗	✗	🟦	✗
LSD-SLAM[23]	Mono	✓	✓	✓	✓	✗	✗	✗	🟦	✓
RGBD-Mapping[24]	RGB-D	✗	✗	✓	✓	✓	✗	✗	🟦	✗
ORB-SLAM2[26]	RGB-D or Stereo or Mono	✗	✗	✓	✓	✓	✓	✗	🟦	✓
ORB-SLAM[27]	Mono	✗	✗	✓	✓	✓	✗	✗	🟦	✓
KinectFusion[28,29]	RGB-D	✓	✓	✓	✗	✓	✗	✓	🟦	✗[d]
VolRTM[30]	RGB-D	✓	✓	✓	✗	✓	✗	✗	🟦	✓
PatchVol[31]	RGB-D	✗	✓	✓	✓	✓	✗	✓	🟦	✗
Kintinuous[32,33]	RGB-D	✓	✓	✓	✗	✓	✗	✓	🟦	✓
HDRFusion[35]	RGB-D	✓	✓	✓	✗	✓	✗	✓	🟦	✓
ElasticFusion[37]	RGB-D	✓	✓	✓	✗	✓	✗	✓	🟦	✓
BundleFusion[38]	RGB-D	✓	✓	✓	✗	✓	✗	✓	🟦	✓

[a] There are forked repositories, but the original has been removed.

[b] Only image regions with strong gradient in the motion direction are used.

[c] Semi-direct: direct methods are used for motion estimation, but features are extracted for keyframes and used for bundle adjustment.

[d] KinectFusion does not seem to have the original MS code published, but there is at least one open source implementation available through the Point Cloud Library (PCL).

2.4 Challenges

These sections have covered the state of the art in ego-motion estimation, and mapping algorithms for a variety of tasks. Some of them perform very well in the environments (or datasets) they are tested in, but our primary concern is how they will perform in the context of a continuous object learning system. Our view of COLs place them along with other “life-long” learning systems; these are systems that are meant to learn incrementally and adapt online over long periods of time. How will ego-motion estimation and mapping scale to handle experiences that range over larger and larger distances and more complicated, unstructured terrain? This section highlights some of the challenges we see when considering the integration of ego-motion estimation for a continuous object learner.

Many existing ego-motion algorithms can run continuously, specifically those based on visual odometry, providing information about motion within the local environment. However, the longer they run, the larger the pose error relative to ground truth will be without measuring:q against some external source. We take the position that objects are *not* independent of their context (i.e., where they are, how they are used, and their relationship with other nearby objects). Therefore, it is important to place objects within an environmental context so they can be reobserved with confidence. Accurate local metric representations can provide this confidence, and visual odometry methods without this kind of mapping capability may not suffice.

SLAM, on the other hand, may provide the facilities needed to produce accurate local maps (see examples of this in section 2). However, many of these algorithms operate indoors (due to sensor or memory limitations) or on city streets; while they may be cluttered and complex, there is still significant structure in the environment. Army systems meant to operate in a variety of environments may need to adapt to *unstructured* environments (jungles or forests), or environments with few obvious features (like underground tunnels and sewers). Since we have seen no evidence of SLAM being evaluated in arbitrary environments, it is hard to predict how these algorithms will generalize to these situations.

From the perspective of a life-long COL, we predict things will go one of 2 ways: long-term mapping and localization (1) with accurate local metric representations, or (2) without accurate local metric maps. It is relatively clear how objects can be contextually represented in the case of accurate local maps (1), but we then face the following challenges: How does a robot recognize places when they are not exactly the same as the original observation? How can the map adapt over time to changes in object position and lighting? How can these local maps be effectively stored and retrieved? Long-term localization and mapping is a field in itself,[39–46] and while it is outside the overall scope of our current work, it is relevant to the effective functioning of future embodied systems aiming for long-term operation in many different environments.

On the other hand, it may be possible to represent objects in context without representing them in a larger metric space, as in case (2). In this case, relative spatial relationships could be stored for object instances, while the global pose of the instance itself is much less important (or unknown). This implies that object instance *recognition* would become more important than ego-motion estimation over longer time periods, since it would be required to aid in localization. This approach is more closely related to human cognitive mapping, which focuses on significant landmarks and their inter-relationships over direct metric representations. Unfortunately, many existing robotic algorithms assume metric representations for planning, and combining the capabilities of topological and metric mapping to simultaneously support a continuous object learner and existing planning algorithms is a new research area.

Recognizing and handling dynamic objects and environments are particularly important for COLs. Unfortunately, many VO and SLAM algorithms do not support dynamic objects effectively. For example, KinectFusion and other TSDF-based algorithms rely on the stability of surfaces in the map volume to integrate enough values to represent the surface. A dynamic object moving through the map volume will therefore not create a surface. DynamicFusion[47] does handle dynamic objects, but ignores the environment in the process. It is not immediately clear how to generate and represent a static map with dynamic objects, but it will probably rely on a careful combination of the existing approaches and a flexible world representation.

3. Brief Survey: Tracking

In this section, we review the main approaches to tracking objects using visual sensors. We survey motion-based tracking, model-free tracking, and for completeness, model-based tracking. Following the surveys, we provide a break-down of the attributes distinguishing selected algorithms and then discuss the challenges we may face when integrating object tracking into a continuous object learner.

3.1 Motion-Based Tracking

One of the most intuitive methods to identify objects to track is motion detection. For a mobile robot, the problem is more difficult since motion detection for external objects must occur while the robot itself is in motion; therefore, we expect ego-motion estimation to be combined with motion-based object-tracking for online learning systems.

In one of the few works to address both ego-motion and tracking, Wang et al.[4] jointly estimate sensor motion and dynamic object motion in a Bayes filter framework using an Extended Kalman Filter. Since the application is target tracking for autonomous driving applications using a planar 2D laser scanner, tracking and motion estimates are members of the SE(2) group (thus assuming planar motion). The state is defined as $\mathbf{x} = [\mathbf{x_s}, \mathbf{x_T}, \mathbf{x_b}, \mathbf{x_p}, \mathbf{x_C}]$, where $\mathbf{x_s}$ is the sensor pose in fixed (world) frame coordinates, $\mathbf{x_T}$ is the collection of motion states for each currently tracked object in world frame coordinates, $\mathbf{x_b}$ is the collection of 2D laser scan points representing the static background model in world frame coordinates, $\mathbf{x_p}$ is the set of collections of points representing each dynamic tracked object in the object's reference frame, and $\mathbf{x_C}$ is the transform between the sensor and vehicle frame. Target tracks are either added to or subtracted from the state dynamically depending on how well new laser data agree with the target's predicted location. Data association takes place by segmenting the laser scans into distinct clusters, then labeling the cluster as static background or dynamic object using ICP. Due to the simplified SE(2) assumptions, the tracker is robust and is capable of dynamically tracking multiple moving objects, but further work would be required to extend the tracking algorithm 3D point clouds and SE(3) motion.

Ondruska et al.[48,49] use a recurrent neural network (RNN) to both track and classify objects in SE(2). The approach uses motion cues to segment, track, and label moving objects in the environment with a static 2D laser scanner. The algorithm is trained to predict the values of a future laser scan x_{t+1} given previous data $x_1, x_2, \ldots, x_t$. While the input to the algorithm is only a single occupancy grid computed from the laser scan, the RNN implicitly learns to model historical dependencies in the input data. This allows for unsupervised training, since the learning signal is just the difference between the predicted laser scan values and the actual values encountered at time $t + 1$. The approach can even estimate a track through a full occlusion in contrast with many tracking algorithms that are unable to continue an existing track after the target disappears from view. The algorithm was tested using static laser scanners, thus limiting its value to a mobile autonomous system, and it is unknown whether such performance could be learned with a similar approach on a mobile platform.

Held et al.[50,51] present an algorithm using annealed dynamic histograms to achieve anytime tracking of dynamic objects using depth, color, and inertial data. Anytime tracking guarantees the availability of a tracking estimate on-demand by sampling a search space in a coarse to fine fashion during optimization, allowing the state of a live mission to determine when to request (or demand) an update. The algorithm samples the parameter space with incrementally increasing resolution for optimization, finding progressively more accurate estimates over time. The simulated annealing algorithm, illustrated in Fig. 8, is a heuristic-based global search that attempts to find the best approximate global extremum versus a precise local extremum. Thus, the expectation in this algorithm is that annealing will provide a coarse estimate of the *global* optimum rather than a finer estimate at a *local* optimum.

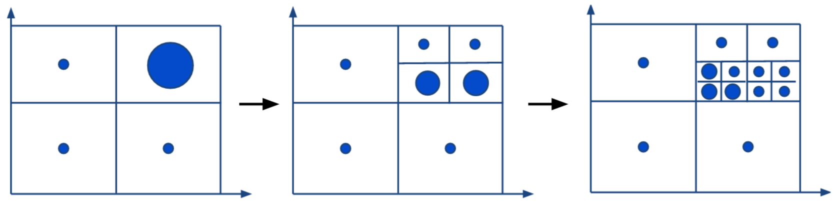


Fig. 8 Visual representation of simulated annealing algorithm for anytime optimization. From left to right, the parameter grid is first coarsely sampled and quickly optimized, followed by optimizations with successively higher resolution sampling to improve accuracy. Reproduced from Held et al.[51]

3.2 Model-Free Tracking

Object tracking using visual sensors, primarily camera sensors, is a well-studied field, as Smeulders et al.[52] describe in a recent survey. As a result of the community's interest in visual object tracking, competitions are held each year to identify the most accurate and robust tracking implementations.

Over recent competitions organized by Kristan et al.,[53,54] the highest-performing model-free algorithms are either based on discriminative correlation filters (DCFs) or deep neural networks (DNNs)—in particular convolutional neural networks (CNNs) or RNNs.

Bolme et al.[55] first propose using DCF to train a simple detector online with few training examples, using an efficient approach for image correlation. The DCF algorithm learns to classify a target from the background by training a sliding window filter to identify the location of the maximum correlation between the input and filter. Algorithmic efficiency is achieved by leveraging the relationship between circular correlation and discrete Fourier transforms (DFTs), which is given as

$$\mathcal{F}(f \star h) = \mathcal{F}(f) \odot \mathcal{F}^*(h), \tag{3}$$

where $\mathcal{F}^*(h)$ denotes the complex conjugate of the Fourier transform of filter h, $f \star h$ is the circular correlation between input f and filter h, and $\mathcal{F}(f) \odot \mathcal{F}^*(h)$ is the element-wise multiplication of the Fourier transforms. The filter is initialized from a set of training patches in the first frame of a tracking sequence, where the training patches include the labeled target region as well as random affine transformations of the target region. In subsequent frames, the algorithm estimates the location of the target as that which maximizes the correlation, then updates the filter (often with weighted averaging) to include the new appearance of the target. The optimization is formulated as

$$\min_{\hat{h}^*} \sum_i |\hat{f}_i \odot \hat{h}^* - \hat{g}_i|^2, \tag{4}$$

where $\hat{f}_i := \mathcal{F}(f_i)$, the circumflex (or hat) denotes the Fourier transform of a function, subscript i indicates the training sample, and g_i is the goal output that is required to generate the training signal. The form of g_i is a design choice, and is typically selected to be a compact 2D Gaussian with a peak centered on the tracking target. The final formulation of the DCF reduces to

$$\hat{h}_i^* = \frac{\hat{g} \odot \hat{f}_i^*}{\hat{f}_i \odot \hat{f}_i^* + \lambda}, \tag{5}$$

where λ is a regularization term.

Since DCF-based algorithms formulate the problem as a circular correlation between filter and input, they are fundamentally susceptible to horizontal and vertical aliasing. To account for this problem, Bolme et al.[55] use a cosine window to preprocess input patches to emphasize locations near the center of the target. Danelljan et al.[56] redefine the optimization formulation to replace the original regularization term with a more general Tikhonov regularization. The regularization weights determine the importance of the filter coefficients *depending on the filter's spatial location*. Filter coefficients far outside the target region are penalized and thus also emphasize locations near the center of the target. Fernandez et al.[57] perform zero-padding on the input and force the larger filter tail to be zero during training, thereby effectively converting the problem from one of circular convolution to one of linear convolution while still employing the DFT for optimization.

Additional work has generalized the DCF formulation to support learning multidimensional filters from multidimensional input descriptors, such as histogram of oriented gradientss (HOGs),[58] color statistics,[59] and CNN features.[60,61] To support target scale change over time, which is a problem for traditional DCF approaches, algorithms can perform regression on bounding box corner points[62,63] or can learn scale explicitly.[64,65] The creation of Kernelized Correlation Filters (KCFs)[66] demonstrates that, by representing the input using circulant matrices, kernel functions can be used to improve training performance without any additional runtime cost, and that the kernel-based problem formulation yields the same analytic solution as the original DCF formulation. Finally, Danelljan et al.[67] present a correlation filter extension that implicitly learns a continuous convolution operator, thereby allowing them to perform DCF operations on multiresolution feature maps (e.g., the outputs of various layers of a CNN).

Inspired by their recent success in object detection, deep networks have become popular approaches to visual object tracking. CNNs are known to produce increasingly discriminative semantic representations at deeper network layers; however, the receptive fields at these layers are also large, which may harm localization accuracy. Since tracking is largely focused on *target* localization accuracy, lower layers are often employed due their smaller receptive field sizes.[60,68] To learn general, class-agnostic similarity functions, siamese networks are trained on consecutive frames for single and multiobject tracking.[69,70]

Held et al.[63] train a CNN on pairs of consecutive images in a sequence, using data augmentation to train on large single-image datasets such as ImageNet. The network implicitly learns both the output tracking bounding box and a motion model that includes scale. To show adequate generalization performance, the authors use disjoint training and test datasets.

The tree-based CNN approach of Nam et al.[71] mitigates the catastrophic forgetting problem of traditional deep networks[72] by maintaining multiple CNNs in a tree structure. The CNNs can be run independently or chained together, so that any newly created CNNs do not actually overwrite previously learned parameters of existing CNN.

Multidomain networks (MDNets)[73] pose the tracking problem as learning across a set of distinct domains, where each tracking sequence in the training set represents a separate domain. The resulting architecture learns a standard CNN that produces an encoded representation of the input image. The output parameters are copied as input into a fully connected layer made up of k parallel domain branches, where k is the number of training sequences. Each sequence is trained exclusively during a single iteration of Stochastic Gradient Descent. During testing, the domain branches are discarded and replaced with a single branch representing the test domain, which is then fine-tuned online. A diagram of the architecture is shown in Fig. 9.

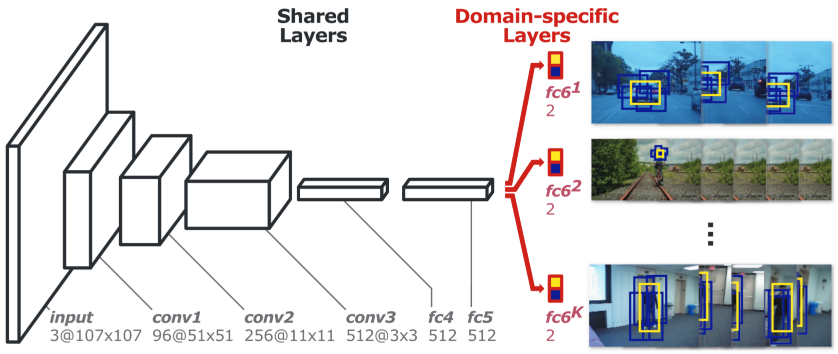


Fig. 9 Architecture diagram for multidomain deep networks for tracking. Each training sequence represents a different domain, with associated parameters $\mathrm{fc6}^1$ **to** $\mathrm{fc6}^k$ **for** k **sequences. Reproduced from Nam et al.[73]**

Valmadre et al.[74] extend the unification of DCF and DNN to produce a fully integrated deep network, shown in Fig. 10, that exhibits complementary the benefits of both DCF and DNN. Instead of using CNN features as input to a DCF framework, as various approaches have recently done, they interpret the correlation filter as a differentiable function that allows it to be inserted as a layer component in a deep network. The value is that the network and associated parameters, including those of the DCF, are jointly learned for tracking. Using a siamese network, the authors produce a tracker that runs at a high frame rate and achieves good accuracy with a relatively shallow network of only 2-3 layers.

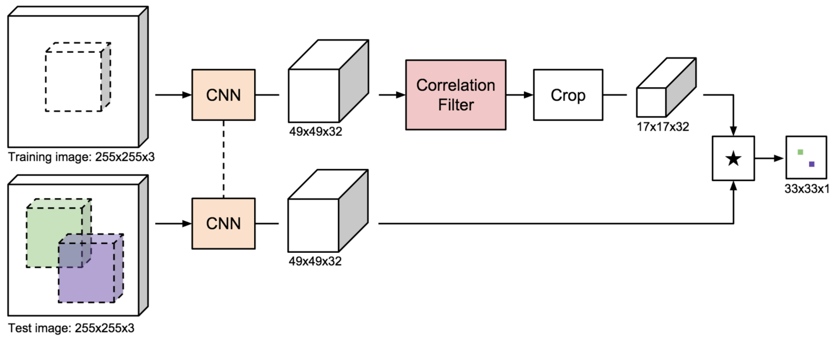


Fig. 10 Integration of a correlation filter layer into a deep network architecture. The correlation filter provides fast tracking evaluation over a sample image, while the deep architecture allows all parameters to be learned jointly. Reproduced from Valmadre et al.[74]

Depth is an important modality for tracking as well as segmentation, since it can directly measure target scale. Song et al.[75] create a new tracking dataset that can be used to test RGB, RGB-D, or 3D tracking algorithms, and provide both RGB-D and 3D baseline algorithms with occlusion handling. Hannuna et al.[76] extend the KCF to include depth, leveraging depth to detect occlusion and scale changes to improve performance while still running at real time. Li et al.[77] present a model-free tracker using RGB-D input data. The algorithm does not require human input to determine the objects to track; instead, it performs a 3D cluster-based segmentation to estimate the objects, thus tracking is performed on *all* segments in a simple tabletop scene.

3.3 Model-Based Tracking

While model-based tracking can be applied to both single and multiobject tracking, we will focus on recent multiobject trackers that make use of pre-trained object models. While model-based trackers have the drawback of requiring known objects ahead of time, they have practical advantages over model-free trackers, such as the ability to dynamically create and destroy object tracks. The challenge of multiobject tracking is to model all object tracks simultaneously, requiring disambiguation as multiple objects interact with overlapping tracks. As with single object tracking, multiple-object tracking is an active research area (see Luo et al.[78] for a review), with benchmark competitions such as the multiobject tracking challenge (MOT[79]) enabling direct comparisons.*

Originally proposed by Reid et al.,[81] Multiple Hypothesis Tracking (MHT) is an algorithm that creates a series of trees representing potential tracks for each object. An advantage of this approach is that the trees can recover from temporarily incorrect tracks by preserving multiple hypothesized states over time. As new information is accumulated, better track estimates can be achieved using the history of previous frames. The level of each tree corresponds to a frame, and since each frame can add multiple potential states to a tree, the trees tend to grow very large without aggressive pruning techniques. Also, at the time of the original publication, poor object detectors and descriptors caused unreliable track estimates. Kim et al.[82] achieve state-of-the-art results using the Maximum Weighted Independent Set method to determine final track hypotheses, and CNN descriptors for appearance similarity scores.

Sadeghian et al.[70] construct a stacked series of RNNs to model temporal dependencies across multiple cues. Each cue (appearance, motion, interaction) is itself an RNN, specifically a long-short-term memory (LSTM) network. LSTM[83,84] networks are an extension of a typical RNN designed to account for the vanishing gradient problem that plagued early multilayer RNNs,[85] yet they are still able to learn long-term dependencies using explicit gates that control how quickly and often previous information is forgotten. The outputs of the cue networks are concatenated and input to another LSTM that outputs the similarity between previous known targets to all current detections. During an ablation study, the appearance cues (where appearance is encoded using CNN descriptors which are input to the LSTM) are shown to be the most important, while the combination of the 3 cues gives the best overall performance.

Similarly, Milan et al.[80] employ a sequence of RNNs to efficiently perform multi-object tracking. The tracking problem is decoupled into prediction/update phases, where prediction incorporates the motion model and update performs data association over time. While simple RNNs can perform the prediction component, the data association and state update are performed using a series of LSTMs, where each

*As pointed out by Milan et al.,[80] an issue with the MOT competition is that authors can either use predefined or custom object detectors. This effectively renders direct comparison of the *tracking algorithms* meaningless, since performance could be affected more by custom object detectors than the tracking algorithms themselves. Instead, only those algorithms using predetected objects can be meaningfully compared.

output encodes the probability of an existing target matching a particular object in the current frame. The architecture diagram of the system is shown in Fig. 11.

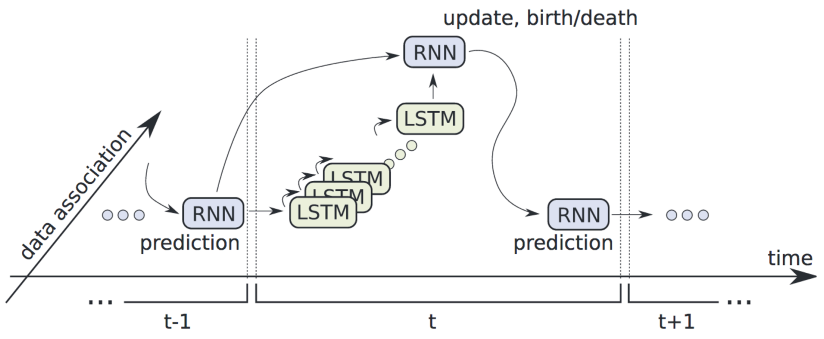


Fig. 11 Architectural diagram for multi-object tracking using RNNs and stacked LSTMs. The RNNs are used to predict and manage tracks over time, while a sequence of LSTMs are used to perform data association between current observations and previously tracked objects. Reproduced from Milan et al.[80]

Levinkov et al.[86] develop a more general approach to 2 problems that are pervasive in computer vision. The first is the graph decomposition problem, where a graph is segmented into subgraphs such that each subgraph is connected. The second problem is that of node labeling, or assigning a class label to each node in a graph. These problems often occur together in practice, and the authors develop an algorithm called the minimum cost node labeling lifted multicut problem (NL-LMP), which generalizes separate approaches to each problem into a single framework. The authors analyze performance on the tasks of articulated human body pose estimation, multi-instance semantic segmentation, and multiobject tracking, where they demonstrate comparable performance with other state-of-the-art algorithms.

In 3D, Ren et al.[87,88] create a multiobject tracker where models are specified with signed distance function (SDF). While being a model-based tracking implementation, the algorithm also adapts the appearance model online to account for the specific object instance being tracked. The technique uses a generative probabilistic model that incorporates the SDF and color data with a latent variable determining if the input sample is a foreground or background instance, and can disambiguate targets of the same class occluding one another.

Choi et al.[89] employ particle filters to estimate the pose of known objects using RGB-D sensors. The tracking problem is posed as the sequential determination of the poses of known objects, where the poses are represented as weighted particles.

The representation combines the point coordinates of the object as well as the color and local normal estimates. Using GPU acceleration, the authors show accuracy and speed improvements (for less than 3200 particles) over a comparable Point Cloud Library (PCL)[90] tracking implementation.

3.4 Attributes

In this subsection, we identify salient properties of the algorithms reviewed in the object tracking survey. Table 2 compares selected algorithms according to the following attributes:

Type High-level approach ([Model-based], [Model-free], [Motion-based])

Initialization Initial target identification source ([Manual], [Motion], [Clustering], [Detector])

Input What is the input sensing modality? ([Mono]cular image, [RGB-D], [3D] LIDAR, [2D] LIDAR)

Representation How is the raw input represented for a tracking algorithm? ([HOG], [CNN], [Raw] input, [Motion], [3D] features, [Normals], etc.)

Algorithm Algorithmic approach ([CNN], [RNN], [Siamese] network, [Corr]elation filter, [Bayes] filter, etc.)

Multiobject Can the algorithm track multiple objects simultaneously?

Scale-adaptive Does the tracking boundary dynamically update scale and aspect ratio?

Code Do the authors make source code available?

Efficiency Approximate estimate of relative efficiency of algorithm.

Table 2 An overview of selected tracking algorithms reviewed in this section and their attributes.

	Type	Initialization	Input	Representation	Algorithm	Multi-Object	Scale-Adaptive	Code	Efficiency
GOTURN[63]	Model-free	Manual	Mono	CNN	CNN	✗	✓	✓	🟦
KCF[66]	Model-free	Manual	Mono	HOG	Corr	✗	✗	✓	🟦
SRDCF[56,60]	Model-free	Manual	Mono	HOG/CNN[a]	Corr	✗	✓	✓	🟦
DS-KCF[76]	Model-free	Manual	RGB-D	HOG+Depth	Corr	✗	✓	✓	🟦
TCNN[71]	Model-free	Manual	Mono	CNN	CNN	✗	✓	✓	🟦
MOSSE[55]	Model-free	Manual	Mono	Raw pixels	Corr	✗	✗	✓	🟦
Staple[59]	Model-free	Manual	Mono	HOG+ColorHist	Hybrid[b]	✗	✓	✓	🟦
Anytime3D[50,51]	Motion-based	Clustering	Mono+3D	Color+3D+Motion	Bayes	✓	✓	✓	🟦
MDNet[73]	Model-free	Manual	Mono	CNN	CNN	✗	✓	✓	🟦
C-COT[67]	Model-free	Manual	Mono	CNN	Corr	✗	✓	✓	🟦
SiamFC[69]	Model-free	Manual	Mono	CNN	Siamese	✗	✓	✓	🟦
PMOT[77]	Model-free	Clustering	RGB-D	JCSD[c]	Bayes	✓	✓	✓	🟦
RGBDOcc[75]	Model-free	Manual	RGB-D	HOG/3D	Various[d]	✗	✓	✓	🟦
DynTrack[4]	Motion-based	Motion	2D	Raw scans+Motion	Bayes	✓	✓	✗	🟦
RNNScan[48,49]	Motion-based	Motion	2D	Raw scans	RNN	✓	✓	✓	🟦
CFNet[74]	Model-free	Manual	Mono	CNN	Siamese + Corr	✓	✓	✓	🟦
MHT[82]	Model-based	Detector	Mono	CNN	Multi-HypTree	✓	✓	✓	🟦
CueRNN[70]	Model-based	Detector	Mono	CNN+Motion+Interaction	RNN	✓	✓	✗	🟦
MTT-RNN[80]	Model-based	Detector	Mono	LSTM	RNN	✓	✓	✓	🟦
NL-LMP[86]	Model-based	Detector	Mono	Detection locations	Lifted MultiCut	✓	✓	✓	🟦
SDF-Track[87,88]	Model-based	Detector	RGB-D	3D SDF	SDF Matching	✓	✓	✓	🟦
RGBD-PF[89]	Model-based	Detector	RGB-D	Color+3D+Normals	Bayes	✓	✓	✗	🟦

[a] The original SRDCF work employs HOG features, the subsequent formulation uses CNN features.

[b] Two independent optimizations occur, one using correlation filters, the other using linear regression.

[c] Joint color-spatial descriptor, approximates the probability density of a point in a 3D+color space.

[d] The focus of the paper is the creation of an RGB-D dataset; the authors develop various 2D and 3D baseline algorithms.

3.5 Challenges

We assert that the design of a continuous object learner implies the use of a primitive object tracker (i.e., a model-free or motion-based tracker), since we do not know about all possible object classes ahead of time. However, model-based object trackers that are trained for specific object classes tend to have better performance when tracking known object classes. The challenge is how to improve the performance of the primitive trackers without incorporating class-specific information. On the other hand, it may be possible to merge primitive tracking techniques with model-based trackers; the challenge in this case includes not only the communication between the algorithms, but how one can scale the performance of a model-based tracker to the many classes a COL may be expected to learn.

When considering model-free trackers, manual object initialization is not a realistic assumption for an autonomous mobile robot. Instead, object motion is a natural choice for a category-agnostic tracking cue. For sensors that are still or do not move much, as is the case with most visual object tracking datasets, it is fairly straightforward to identify moving objects. For a mobile system, however, the sensor will move throughout the scene, with potentially large changes in perspective making technique like background substraction infeasible. Therefore, algorithms must be able to model either ego-motion (or receive it from an estimation algorithm) or the static and dynamic environment components, which are complementary tasks. This poses challenges for otherwise high-performing approaches trained on static or nearly static sensor streams, such as most discriminative correlation filters or deep learning approaches.

Finally, tracking multiple objects poses its own set of challenges. One may construct either a single model capable of tracking all moving objects, or dynamically instantiate multiple single-object trackers. In either case, trackers must handle occlusions, ideally even total occlusions, and must resolve ambiguity when multiple tracked targets interact or cross paths. While existing trackers struggle with these challenges, a successful multiobject tracking algorithm will improve the performance of downstream components such as instance recognition.

4. Conclusion

In the Introduction, we argue that ego-motion estimation and object tracking are essential components for a continuous object learner. They provide continuity of the incoming sensor data in order to make sense of the world; otherwise it would be very difficult to relate sensor data at one time instant to data at another time. Having briefly surveyed state-of-the-art algorithms in VO, SLAM, and several approaches to object tracking, we then discussed the challenges of integrating these into a continuous object learning system.

A continuous object learner will need to initially tackle the hard problems of integrating tracking systems with ego-motion to better handle dynamic environments; this is a requirement to learn from experience in the real world, and it has not been thoroughly addressed in the research literature thus far. Enabling longer-term operation, especially in novel, unstructured environments, requires research in scaling maps or even operating without the traditional metric notion of a map. In either case, we must also research methods of supporting contextual object representation within these maps.

Finally, building a full continuous object learning system should provide opportunities to leverage capabilities across various components; however, this too poses many challenges. For example, how should an ego-motion algorithm work with an object tracking component; can they be de-coupled, or do they need to share data structures? Is it possible that ego-motion estimation or object tracking can benefit from segmentation or instance recognition (and what information should they share) or vice versa? These are just some of the questions that must be addressed in future research toward continuous object learning systems.

5. References

1. Fetsch CR, Turner AH, DeAngelis GC, Angelaki DE. Dynamic reweighting of visual and vestibular cues during self-motion perception. Journal of Neuroscience. 2009;29(49):15601–15612.

2. Koffka K. Principles of Gestalt psychology. New York (NY): Harcourt, Brace, & World; 1935.

3. Zhu G, Porikli F, Li H. Tracking randomly moving objects on edge box proposals. Ithaca (NY): arXiv.org, Cornell University Library; 2015. arXiv: 1507.08085 [cs.CV].

4. Wang DZ, Posner I, Newman P. Model-free detection and tracking of dynamic objects with 2D lidar. The International Journal of Robotics Research. 2015;34(7):1039–1063.

5. Engel J, Koltun V, Cremers D. Direct sparse odometry. Ithaca (NY): arXiv.org, Cornell University Library; 2016. arXiv: 1607.02565.

6. Lowe DG. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision. 2004;60(2):91–110.

7. Rublee E, Rabaud V, Konolige K, Bradski G. ORB: an efficient alternative to SIFT or SURF. In: Computer Vision (ICCV), 2011 IEEE International Conference on; p. 2564–2571.

8. Geiger A, Ziegler J, Stiller C. Stereoscan: dense 3D reconstruction in realtime. In: Intelligent Vehicles Symposium (IV), 2011 IEEE; p. 963–968.

9. Geiger A, Lenz P, Urtasun R. Are we ready for autonomous driving? The KITTI vision benchmark suite. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition; p. 3354–3361.

10. Scaramuzza D, Fraundorfer F. Visual Odometry [Tutorial]. IEEE Robotics & Automation Magazine. 2011;18(4):80–92.

11. Steinbrucker F, Sturm J, Cremers D. Real-time visual odometry from dense RGB-D images. In: Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on; p. 719–722.

12. Zhang J, Singh S. LOAM: Lidar odometry and mapping in realtime. In: Robotics: Science and Systems Conference (RSS 2014).

13. Zhang J, Singh S. Visual-lidar odometry and mapping: low-drift, robust, and fast. In: 2015 IEEE International Conference on Robotics and Automation (ICRA); IEEE; 2015. p. 2174–2181.

14. Velodyne LiDAR I. 2017 [accessed 2017 Sep 22]. http://velodynelidar.com/.

15. Zhang J. loam_continuous - ROS Wiki. 2017 [accessed 2017 Sep 22]. http://wiki.ros.org/loam_continuous.

16. Dunkley O, Engel J, Sturm J, Cremers D. Visual-inertial navigation for a camera-equipped 25g nano-quadrotor. In: IROS2014 aerial open source robotics workshop; p. 2.

17. Schöps T, Engel J, Cremers D. Semi-dense visual odometry for AR on a smartphone. In: Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on; IEEE; 2014. p. 145–150.

18. Engel J, Sturm J, Cremers D. Semi-dense visual odometry for a monocular camera. In: The IEEE International Conference on Computer Vision (ICCV); 2013.

19. Forster C, Pizzoli M, Scaramuzza D. SVO: fast semi-direct monocular visual odometry. In: Proc IEEE Intl Conf on Robotics and Automation; p. 15–22.

20. Weiss S. Dealing with scale; 2014 [accessed 2017 Sep 20]. http://frc.ri.cmu.edu/~kaess/vslam_cvpr14/media/VSLAM-Tutorial-CVPR14-A21-DealingWithScale.pdf.

21. Cadena C, Carlone L, Carrillo H, Latif Y, Scaramuzza D, Neira J, Reid I, Leonard JJ. Past, present, and future of simultaneous localization and mapping: toward the robust-perception age. IEEE Transactions on Robotics. 2016;32(6):1309–1332.

22. Engel J, Stückler J, Cremers D. Large-scale direct slam with stereo cameras. In: Proceedings of the 2015 IEEE International Conference on Intelligent Robots and Systems (IROS); 2015. p. 1935–1942.

23. Engel J, Schöps T, Cremers D. LSD-SLAM: large-scale direct monocular SLAM. In: Fleet D, Pajdla T, Schiele B, Tuytelaars T, editors. Computer Vision–ECCV 2014; Cham (Switzerland): Springer; 2014. p. 834–849. (Lecture notes in computer science; vol. 8690).

24. Henry P, Krainin M, Herbst E, Ren X, Fox D. RGB-D mapping: using kinect-style depth cameras for dense 3d modeling of indoor environments. The International Journal of Robotics Research. 2012;31(5):647–663.

25. Pfister H, Zwicker M, van Baar J, Gross M. Surfels: surface elements as rendering primitives. In: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques; (SIGGRAPH ’00) New York (NY): ACM Press; 2000. p. 335–342.

26. Mur-Artal R, Montiel JMM, Tardos JD. ORB-SLAM: a versatile and accurate monocular SLAM system. IEEE Transactions on Robotics. 2015;31(5):1147–1163.

27. Mur-Artal R, Tardos JD. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. Ithaca (NY): arXiv.org, Cornell University Library; 2016. arXiv:1610.06475.

28. Newcombe RA, Izadi S, Hilliges O, Molyneaux D, Kim D, Davison AJ, Kohli P, Shotton J, Hodges S, Fitzgibbon A. KinectFusion: Real-time dense surface mapping and tracking. In: Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on; p. 127–136.

29. Izadi S, Kim D, Hilliges O, Molyneaux D, Newcombe R, Kohli P, Shotton J, Hodges S, Freeman D, Davison A, et al. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In: Proceedings of the 24th annual ACM symposium on User interface software and technology; New York (NY): ACM; 2011. p. 559–568.

30. Steinbrücker F, Sturm J, Cremers D. Volumetric 3D mapping in real-time on a CPU. In: Robotics and Automation (ICRA), 2014 IEEE International Conference on; IEEE; 2014. p. 2021–2028.

31. Henry P, Fox D, Bhowmik A, Mongia R. Patch volumes: segmentation-based consistent mapping with RGB-D cameras. In: 2013 International Conference on 3D Vision - 3DV; IEEE; 2013. p. 398–405.

32. Whelan T, McDonald J, Kaess M, Fallon M, Johannsson H, Leonard J. Kintinuous: Spatially extended kinect fusion. In: Robotics, Science and Systems.

33. Whelan T, Johannsson H, Kaess M, Leonard JJ, McDonald J. Robust tracking for real-time dense RGB-D mapping with kintinuous. DSpace@MIT; 2012. Technical No.: MIT-CSAIL-TR-2012-031 [accessed 2017 June 28]. http://hdl.handle.net/1721.1/73167.

34. Whelan T, Kaess M, Johannsson H, Fallon M, Leonard JJ, McDonald J. Real-time large-scale dense RGB-D SLAM with volumetric fusion. The International Journal of Robotics Research. 2015;34(4-5):598–626.

35. Li S, Handa A, Zhang Y, Calway A. HDRFusion: HDR SLAM using a low-cost auto-exposure RGB-D sensor. Ithaca (NY): arXiv.org, Cornell University Library; 2016. arXiv:1604.00895.

36. Lee SO, Lim H, Kim HG, Ahn SC. RGB-D fusion: Real-time robust tracking and dense mapping with RGB-D data fusion. In: Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on; IEEE; 2014. p. 2749–2754.

37. Whelan T, Leutenegger S, Salas-Moreno RF, Glocker B, Davison AJ. ElasticFusion: Dense SLAM without a pose graph. Proc Robotics: Science and Systems; 2015; Rome, Italy.

38. Dai A, Nießner M, Zollhöfer M, Izadi S, Theobalt C. BundleFusion: Real-time globally consistent 3D reconstruction using on-the-fly surface re-integration. Ithaca (NY): arXiv.org, Cornell University Library; 2016. arXiv:1604.01093.

39. Konolige K, Bowman J. Towards lifelong visual maps. In: Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on; IEEE; 2009. p. 1156–1163.

40. Biswas J, Veloso MM. Localization and navigation of the cobots over long-term deployments. The International Journal of Robotics Research. 2013;32(14):1679–1694.

41. Churchill W, Newman P. Experience-based navigation for long-term localisation. The International Journal of Robotics Research. 2013;32(14):1645–1661.

42. Johannsson H. Toward lifelong visual localization and mapping [thesis]. [Cambridge (MA)]: Massachusetts Institute of Technology; 2013.

43. Tipaldi GD, Meyer-Delius D, Burgard W. Lifelong localization in changing environments. The International Journal of Robotics Research. 2013;32(14):1662–1678.

44. Beall C, Dellaert F. Appearance-based localization across seasons in a metric map. 6th PPNIV; 2014; Chicago, IL.

45. Mühlfellner P. Lifelong visual localization for automated vehicles. [PhD thesis]. [Sweden]: Halmstad University; 2015.

46. Krajnik T, Pulido Fentanes J, Hanheide M, Duckett T. Persistent localization and life-long mapping in changing environments using the frequency map enhancement. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS); Daejeon, Korea: IEEE; 2016.

47. Newcombe RA, Fox D, Seitz SM. Dynamic fusion: reconstruction and tracking of non-rigid scenes in real-time. In: Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition; 2015. p. 343–352.

48. Ondruska P, Dequaire J, Wang DZ, Posner I. End-to-end tracking and semantic segmentation using recurrent neural networks. Ithaca (NY): arXiv.org, Cornell University Library; 2016. arXiv:1604.05091.

49. Ondruska P, Posner I. Deep tracking: seeing beyond seeing using recurrent neural networks. Ithaca (NY): arXiv.org, Cornell University Library; 2016. arXiv:1602.00991.

50. Held D, Levinson J, Thrun S. Precision tracking with sparse 3D and dense color 2D data. In: Robotics and Automation (ICRA), 2013 IEEE International Conference on; IEEE; 2013. p. 1138–1145.

51. Held D, Levinson J, Thrun S, Savarese S. Combining 3d shape, color, and motion for robust anytime tracking. In: Proceedings of 2014 Robotics: Science and Systems; 2014; Berkeley, CA.

52. Smeulders AWM, Chu DM, Cucchiara R, Calderara S, Dehghan A, Shah M. Visual tracking: an experimental survey. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2014;36(7):1442–1468.

53. Kristan M, Matas J, Leonardis A, Felsberg M, Cehovin L, Fernandez G, Vojir T, Hager G, Nebehay G, Pflugfelder R. The visual object tracking VOT2015 challenge results. In: Proceedings of the IEEE International Conference on Computer Vision Workshops; 2015. p. 1–23.

54. Kristan M, Leonardis A, Matas J, Felsberg M, Pflugfelder R, Cehovin L, Vojir T, Hager G, Lukezic A, Fernandez G. The visual object tracking VOT2016 challenge results. In: Proceedings of the IEEE European Conference on Computer Vision Workshops; 2016.

55. Bolme DS, Beveridge JR, Draper BA, Lui YM. Visual object tracking using adaptive correlation filters. In: Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on; IEEE; 2010. p. 2544–2550.

56. Danelljan M, Hager G, Shahbaz Khan F, Felsberg M. Learning spatially regularized correlation filters for visual tracking. In: Proceedings of the 2015 IEEE International Conference on Computer Vision; 2015. p. 4310–4318.

57. Fernandez JA, Boddeti VN, Rodriguez A, Kumar BVKV. Zero-aliasing correlation filters for object recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2015;37(8):1702–1715; arXiv: 1411.2316.

58. Kiani Galoogahi H, Sim T, Lucey S. Multi-channel correlation filters. In: Proceedings of the 2013 IEEE International Conference on Computer Vision; 2013. p. 3072–3079.

59. Bertinetto L, Valmadre J, Golodetz S, Miksik O, Torr P. Staple: complementary learners for real-time tracking. Ithaca (NY): arXiv.org, Cornell University Library; 2015. arXiv:1512.01355 [cs].

60. Danelljan M, Hager G, Shahbaz Khan F, Felsberg M. Convolutional features for correlation filter based visual tracking. In: Proceedings of the 2015 IEEE International Conference on Computer Vision Workshops; 2015. p. 58–66.

61. Ma C, Huang JB, Yang X, Yang MH. Hierarchical convolutional features for visual tracking. In: Proceedings of the 2015 IEEE International Conference on Computer Vision; 2015. p. 3074–3082.

62. Bertinetto L, Valmadre J, Miksik O, Golodetz S, Torr P. The importance of estimating object extent when tracking with correlation filters. 2015. Pre-print for VOT2015.

63. Held D, Thrun S, Savarese S. Learning to track at 100 FPS with deep regression networks. In: Leibe B, Matas J, Sebe N, Welling M, editors. European Conference on Computer Vision; Cham (Switzerland): Springer; 2016. p. 749–765. (Lecture notes in computer science; vol. 9905).

64. Danelljan M, Hager G, Khan FS, Felsberg M. Discriminative scale space tracking. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2016.

65. Vojir T, Noskova J, Matas J. Robust scale-adaptive mean-shift for tracking. In: Kämäräinen JK, Koskela M, editors. Scandinavian Conference on Image Analysis; Berlin Heidelberg (Germany): Springer; 2013. p. 652–663. (Lecture notes in computer science; vol. 7944).

66. Henriques JF, Caseiro R, Martins P, Batista J. High-speed tracking with kernelized correlation filters. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2015;37(3):583–596.

67. Danelljan M, Robinson A, Khan FS, Felsberg M. Beyond correlation filters: learning continuous convolution operators for visual tracking. In: Leibe B, Matas J, Sebe N, Welling M, editors. European Conference on Computer Vision; Springer; 2016. p. 472–488. (Lecture notes in computer science; vol. 9909). DOI: 10.1007/978-3-319-46454-1_29.

68. Wang L, Ouyang W, Wang X, Lu H. Visual tracking with fully convolutional networks. In: 2015 IEEE International Conference on Computer Vision (ICCV); IEEE; 2015. p. 3119–3127.

69. Bertinetto L, Valmadre J, Henriques JF, Vedaldi A, Torr PHS. Fully-convolutional siamese networks for object tracking. Ithaca (NY): arXiv.org, Cornell University Library; 2016. arXiv:1606.09549 [cs].

70. Sadeghian A, Alahi A, Savarese S. Tracking the untrackable: learning to track multiple cues with long-term dependencies. Ithaca (NY): arXiv.org, Cornell University Library; 2017. arXiv:1701.01909 [cs].

71. Nam H, Baek M, Han B. Modeling and propagating CNNs in a tree structure for visual tracking. Ithaca (NY): arXiv.org, Cornell University Library; 2016. arXiv:1608.07242 [cs].

72. Goodfellow IJ, Mirza M, Xiao D, Courville A, Bengio Y. An empirical investigation of catastrophic forgetting in gradient-based neural networks. Ithaca (NY): arXiv.org, Cornell University Library; 2013. arXiv:1312.6211.

73. Nam H, Han B. Learning multi-domain convolutional neural networks for visual tracking. Ithaca (NY): arXiv.org, Cornell University Library; 2015. arXiv:1510.07945 [cs].

74. Valmadre J, Bertinetto L, Henriques JF, Vedaldi A, Torr PHS. Endto-end representation learning for correlation filter based tracking. Ithaca (NY): arXiv.org, Cornell University Library; 2017. arXiv:1704.06036 [cs].

75. Song S, Xiao J. Tracking revisited using RGBD camera: unified benchmark and baselines. In: Proceedings of the 2013 IEEE international conference on computer vision; 2013. p. 233–240.

76. Hannuna S, Camplani M, Hall J, Mirmehdi M, Damen D, Burghardt T, Paiement A, Tao L. DS-KCF: a real-time tracker for RGB-D data. Journal of Real-Time Image Processing. 2016 [accessed 2017 Sep 19]. https://link.springer.com/article/10.1007%2Fs11554-016-0654-3.

77. Li S, Koo S, Lee D. Real-time and model-free object tracking using particle filter with joint color-spatial descriptor. In: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on; IEEE; 2015. p. 6079–6085.

78. Luo W, Xing J, Zhang X, Zhao X, Kim TK. Multiple object tracking: a literature review. Ithaca (NY): arXiv.org, Cornell University Library; 2014. arXiv:1409.7618 [cs].

79. Milan A, Leal-Taixe L, Reid I, Roth S, Schindler K. MOT16: A benchmark for multi-object tracking. Ithaca (NY): arXiv.org, Cornell University Library; 2016. arXiv:1603.00831 [cs].

80. Milan A, Rezatofighi SH, Dick A, Reid I, Schindler K. Online multi-target tracking using recurrent neural networks. Ithaca (NY): arXiv.org, Cornell University Library; 2016. arXiv:1604.03635.

81. Reid D. An algorithm for tracking multiple targets. In: IEEE Transactions on Automatic Control. 1979 Dec;AC-24(6);843–855.

82. Kim C, Li F, Ciptadi A, Rehg JM. Multiple hypothesis tracking revisited. In: Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV); (ICCV '15)Washington, DC, USA: IEEE Computer Society; 2015. p. 4696–4704.

83. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation. 1997;9:1735–1780.

84. Greff K, Srivastava RK, Koutník J, Steunebrink BR, Schmidhuber J. LSTM: a search space odyssey. Ithaca (NY): arXiv.org, Cornell University Library; 2015. arXiv:1503.04069 [cs].

85. Hochreiter S, Bengio Y, Frasconi P, Schmidhuber J. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. A field guide to dynamical recurrent neural networks. Newark (NJ): IEEE Press; 2001.

86. Levinkov E, Uhrig J, Tang S, Omran M, Insafutdinov E, Kirillov A, Rother C, Brox T, Schiele B, Andres B. Joint graph decomposition and node labeling: problem, algorithms, applications. Ithaca (NY): arXiv.org, Cornell University Library; 2016. arXiv:1611.04399 [cs].

87. Ren CY, Prisacariu V, Kaehler O, Reid I, Murray D. 3D tracking of multiple objects with identical appearance using RGB-D input. In: 2014 2nd International Conference on 3D Vision; IEEE; 2014. p. 47–54.

88. Ren CY, Prisacariu VA, Kähler O, Reid ID, Murray DW. Real-time tracking of single and multiple objects from depth-colour imagery using 3D signed distance functions. International Journal of Computer Vision. 2017;124(1):80–95.

89. Choi C, Christensen HI. RGB-D object tracking: a particle filter approach on GPU. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems; IEEE; 2013. p. 1084–1091.

90. Rusu RB, Cousins S. 3D is here: Point Cloud Library (PCL). In: Robotics and Automation (ICRA), 2011 IEEE International Conference on; IEEE; 2011. p. 1–4.

List of Symbols, Acronyms, and Abbreviations

2D	2-dimensional. 6, 16, 20, 22, 28
3D	3-dimensional. 3, 6–9, 11, 14–16, 20, 25, 27, 28
CNN	convolutional neural network. 22–24, 26, 28
COL	continuous object learner. 1–3, 5, 18, 19, 30
CVPR	Computer Vision and Pattern Recognition. 11
DCF	discriminative correlation filter. 22–25
DFT	discrete Fourier transform. 22, 23
DNN	deep neural network. 22, 24
DSO	Direct Sparse Odometry. 10
DVO	Dense Visual Odometry. iv, 8, 9
GPU	graphics processing unit. 13–16, 28
HOG	histogram of oriented gradients. 23, 28
ICP	iterative closest point. 12, 20
IMU	inertial measurement unit. 3, 11
IR	infrared. 9
KCF	Kernelized Correlation Filter. 23, 25
LIDAR	light detection and ranging. 10, 16, 28
LSTM	long-short-term memory. iv, 26, 27
MDNet	multidomain network. 24
NL-LMP	node labeling lifted multicut problem. 27

ORB oriented FAST and rotated BRIEF. 6, 13

PCL Point Cloud Library. 28

RGB red, green, blue. 12, 25

RGB-D red, green, blue and depth. 6, 8, 9, 11–14, 25, 27

RNN recurrent neural network. iv, 20, 22, 26–28

ROS Robot Operating System. 10

SDF signed distance function. 27

SIFT scale-invariant feature transform. 6, 15

SLAM simultaneous localization and mapping. iv, 4, 11–13, 18, 19, 31

SVO semi-direct visual odometry. 10

TSDF truncated signed distance function. 14, 19

VO visual odometry. 4, 8, 10–13, 19, 31

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO L IMAL HRA MAIL & RECORDS MGMT
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
1 (PDF)	UNIVERSITY OF PENNSYLVANIA DEPT COMPUTER SCIENCE K DANIILIDIS
	<u>ABERDEEN PROVING GROUND</u>
8 (PDF)	DIR USARL RDRL VT A GHOSHAL B SADLER RDRL VTA MA FIELDS H EDGE C KRONINGER J OWENS P OSTEEN RDRL HRF-D T KELLEY

INTENTIONALLY LEFT BLANK.